SearchWinDevelopment.com E-GUIDE

Application Security

The SearchWinDevelopment.com E-Guide: Application Security provides an expert overview of building security into applications during the development process. From the browser to the desktop to mobile applications, security threats are increasing at an alarming rate. Features and functionality are being traded off for security liabilities. This E-Guide analyzes the consequences of these trade-offs and what actions can be taken to limit security threats.

TechTarget
The Technology Media ROI Experts

SearchWinDevelopment.com E-Guide

Application Security

Table of Contents:

Browser security a concern for website development

Jennette Mullaney, Associate Editor

None of the major Web browsers is secure, and without significant changes security will continue to be a problem, according to Web security researchers. Increased functionality, the rise of sophisticated attacks, and a resistance among browser makers to integrate security into product development has left browsers more vulnerable to attacks.

Researchers say that through the addition of features and Web 2.0 technologies, browsers have become more open to attack.

"Back in the days of gopher and static HTML—think before ActiveX controls and JavaScript—the Web was fairly innocuous," said Robert "RSnake" Hansen, CEO of Internet security company SecTheory. "The nature of dynamic Web applications coupled with very complex client-side code has dramatically increased the attack surface area in the browser and ways in which the browser can be used to attack Web applications."

Gary McGraw, CTO of Cigital, agreed that an abundance of features has made browsers less secure. We trade functionality for security, he said.

"The trend is in the wrong direction," he said. "Since Mosaic in 1993, browsers look more like complete operating systems than they did back in those days."

However, it would be difficult to find anyone who would want to trade in Firefox for Mosaic. Hansen half-joked at a recent OWASP conference that the only way to avoid a new clickjacking threat was to run the text-based browser Lynx, although such masochism would likely be exhibited among only the most paranoid of Web users.

Competition among browsers

When Google released its browser, Chrome, in September, it faced stiff competition from Microsoft's Internet Explorer, Mozilla's Firefox, Opera, and Apple Computer's Safari. Consumers have many browsers to choose from, but their options for security are still limited. None of the browsers is particularly secure, according to Hansen.

Jeremiah Grossman, founder and CTO of WhiteHat Security, put it more plainly: "There is no browser security."

The "browser wars" both help and hinder security, researchers say.

"While there aren't that many more browser experts out there than there were two years ago, their understanding of browser nuances has definitely increased tremendously with the advent of the browser wars," Hansen said. "Competition is ultimately good, because it can force browsers to look for differentiators."

Security is one of those differentiators. However, features that ultimately hamper security are among the differentiators as well, continued Hansen.

Testing headaches

Additional browsers also mean additional work for security and testing professionals.

Mike Kelly, director of testing and quality assurance at Interactions, explained that a "combinatorial problem" arises from the presence of so many browsers and their various versions.

"If you have five browsers, each browser has two versions you support, and you need to support a different variety of security settings, then you have a very large number of platforms to test," he said. "If you assume even something as simple as three different security setting configurations, you're looking at around 5 x 2 x 3 tests—or 30 different configurations."

Bolting on security

It may seem surprising that, with all of the security features browsers come with now, they would be considered insecure. But security add-ons can't completely make up for a lack of security in the SDLC.

"Web browser security add-ons don't do anything, particularly," Grossman said.

To mitigate risks online, Grossman uses an older browser for sensitive transactions. An old browser lacks security features, but its obscurity makes it a less-attractive target for attackers, he explained. McGraw uses Opera for similar reasons. "An attack aimed at IE it isn't likely to work against Opera," he said.

Hansen is concerned about architectural flaws in browsers. Familiar security issues such as cross-site scripting (XSS) and cross-site request forgery (CSRF), are architectural-level design flaws, he said. Clickjacking, a threat uncovered recently by Hansen and Grossman, falls under the same category. These flaws are far more difficult to fix than bugs in code.

"Fixing them means breaking legitimate functionality that exists on the Internet," Hansen explained. "No one wants that—so they go unfixed."

Instead, patches are issues and plug-ins are recommended. Hansen said the browsers have bolted on things such as basic and digest authentication, SSL, HTTPOnly, and some other vaguely useful security policies. However, those measures leave browsers only slightly more secure as attacks become dramatically more sophisticated, he said.

Grossman acknowledged that browser makers are in a tough spot when it comes to security.

"In their defense, security is a really hard problem," he said. "They all say they're doing their best. I say it's insecure."

What browser makers and users can do

Security bugs aren't going to disappear, Kelly said, but there is definitely room for browser makers to do more.

"A development team only has months to harden a browser for security exploits while a hacker has years to find new exploits," he said. "I think there is more we can do collaboratively as an industry."

McGraw is more direct. "Browser vendors should practice software security just like everybody else," he said. "In this case, since they have hundreds of millions of users, it's particularly important that they build security in."

Standardized authentication for websites would be a good step toward security, said Hansen. He also predicted that "site security policies that allow websites to tell browsers how to treat their information and what to do once they visit the site is going to be an important future design improvement for browsers."

Hansen recommended that browsers alert users when they are being sent to Intranet 'zones' such as RFC1918 or localhost. "A lot of the more dangerous theoretical attacks are starting to target internal devices from the browser," he explained.

Users should, as always, practice common sense when giving out sensitive information online. Hansen, Grossman, and McGraw all recommended virtualization as a possible safety option, although it isn't practical for individual users. Limiting sensitive transactions to one browser and surfing to another, as Grossman does, helps to compartmentalize risk and is a good alternative for end users.

How to develop secure applications

Ryan Berg, Contributor

Despite the ubiquitous nature of applications—powering everything from our business processes to our cell phones to our cars—and our heavy reliance on them, minimal efforts are put forth to secure those applications. Most developers start working on tightening security in the testing phase. That's often too little, too late. In this tip, we look at why developing secure code is the first and most important step in developing secure applications and offer some advice on how to secure code well.

Software is generally created from a "functionality first" perspective, with quality as a part of the standard software development lifecycle (SDLC) but with security as a distant third (or nonexistent) priority. This is an unfortunate reality. Designing an application is an exercise in meeting a business objective. The application design and development stage is the ideal time to consider how security requirements and business needs intersect. Building security into the SDLC is a sound business decision—there may be a cost in securing your vulnerabilities, but allowing yourself to be exposed to malicious activities has costs as well. Prevention is a more reasonable cost to justify and ultimately a much lower cost for an organization to absorb.

Studies have repeatedly shown that detecting and preventing code flaws early in the software development life cycle leads to significant cost savings. Unfortunately, the path to securing an application too often begins with rigorous testing for vulnerabilities, to ensure the application will not compromise, or allow others to compromise, data privacy and integrity. This is already too late.

Developing secure code must begin during requirement definition and continue throughout design and development, as well as during testing and deployment. If you wait until testing you are almost guaranteed to find insecurities, and all too often, you will not find all of them or even miss the most critical flaws. Secure coding is, admittedly, a cultural shift for many organizations, because it is such a fundamental, nonstrategic area, yet it has the most intrinsic relationship with data privacy and integrity and is the most effective way to verify that the security requirements set forth during design have been met. The best way to ensure code security is through a secure development process that includes source code review and accomplishes three things:

- Consistency: It's important to create consistent processes and policies for a culture of improved security.
- Multidimensional analysis: When it comes to dangerous vulnerabilities, large-scale design flaws can be more dangerous than the individual coding errors that are more traditionally associated with application vulnerability, such as buffer overflows. Fixing individual vulnerabilities will have little effect if data is not encrypted, authentication is weak, or there are open backdoors in an application.
- Mitigation prioritization: When reviewing existing code, developers must identify all vulnerabilities in the code, prioritize and triage those vulnerabilities in the context of the organization, and then remediate the greatest risks first.

Developing secure source code requires vigilance in examining all of the places vulnerabilities may exist, not just

those where we expect them to exist—for example, through penetration testing. Even with the use of automated tools, the development community needs to validate implementation and design practices, including native code and code-reuse practices, and whether or not they could result in vulnerabilities. Along the way, to effectively measure the risk posed by any given application, security analysts or developers should watch especially for the two primary categories of errors:

- Coding errors: These types of "quality-style" defects are usually minute and will usually stand alone when identified and remediation is applied. They are characterized by "loose" program practices such as buffer overflows and call-timing mismatches.

- Design flaws: This category includes security mechanisms that, when defined properly from the outset, can be part of the positive security in an application, as opposed to an area of risk. These include authentication, encryption, the use of insecure external code types, and validation of data input as well as application output. However, if poorly implemented, they can open up the application to just as much risk as a buffer overflow.

Finding these kinds of vulnerabilities in your applications is one part process and nine parts detective work—it is not just about finding a better way to define the need for security in the development process, but about looking at all of the places where vulnerabilities of all types can lurk and identifying the potential risk to your organization if those vulnerabilities were to be exploited.

Paging Sherlock Holmes

The most common approaches to vulnerability detection are manual code review and penetration testing. Each method approaches the analysis in a different way. Manual code reviews can thoroughly analyze an application across a wide matrix of criteria, but are time-consuming and expensive and do not scale to be a part of the development process given the complexity of most software code. Often manual code review is only performed on those areas in an application that are believed to present the greatest risk, leaving large areas of the application wide open. Vulnerabilities have no prejudice, however, and can exist anywhere.

Penetration tests, when automated, are easily repeated, but must by necessity fall at the end of the lifecycle when the application is complete, rather than being a tool employed from the start. Additionally, they have a more narrowly defined set of vulnerabilities than source code analysis, which identifies a broader array of potential vulnerabilities beyond the expected ones.

Automated source code analysis, while a comparatively new tool in the security analyst's detective kit, arms organizations with the ability to evaluate every application—both existing applications as well as code under development—against critical classes of code vulnerabilities, including:

- Security-related functions

- Input/output (I/O) validation and encoding errors

- Error handling and logging vulnerabilities

- Insecure components
- Coding errors

Following the path of security-related issues through the source code of an application can dramatically reduce the vulnerability of the application and the critical data it processes and protects.

Companies today must treat every existing and under-development application as a security risk until it is proven otherwise, simply because of the risk such vulnerabilities can pose to the business. No single tool will be the silver bullet to make all software secure, simply because the breadth of legacy code in use is so vast, but tools do help developers and code reviewers assess applications to quickly identify the most potentially damaging vulnerabilities and triage those applications for remediation. Taking a risk-based approach to remediating the code base, starting with the most critical problems first, is the most effective means to developing secure applications. Companies able to efficiently and effectively integrate this analysis into their software development lifecycle practices will not only improve their own security state but reap substantial business benefits for themselves and all those that rely on their software.

The essentials of Web application threat modeling

Kevin Beaver, CISSP

A critical part of Web application security is mapping out what's at risk—a process called threat modeling. The term "threat" modeling is actually a misnomer. It's more like "vulnerability" or "risk" modeling, since we're technically looking at weaknesses and their consequences—not the actual indication of intent to cause disruption (a threat).

Semantics aside, threat modeling—even at a high level—needs to be on your radar and part of your development process if Web application security is important to your business. Think about it. There's a lot happening within your Web applications that you may not be aware of. It's really easy to fall into the trap of assuming all's well in Web-land as long as the basics of a firewall, SSL, and strong passwords are in place. This dangerous assumption boils down to not really knowing what's at risk. It's the bane of information security today.

Let threat modeling help fill the gaps. It really does work. Here are the essential steps for getting started:

1. **Determine your security goals** by outlining what's in scope, what's absolutely critical, as well as what's being mandated by management, security policy, or even customers or business associates. Make sure everyone's on the same page and expectations are properly set.

2. **Document the general architecture of your application** and outline information flows from user to Web server, Web server to application server, application server to database server, and so on.

3. **Outline what really needs to be protected** such as user login credentials, session information, source code, application logic, and (especially) customer information.

4. **Pinpoint the various entry points and "trust" zones** that need protection, including user authentication, user management, system logging, server/application maintenance, and critical application and database interfaces.

5. **Discover what can be exploited using a malicious mindset**—from both the perspective of an untrusted outsider and a trusted user.

   You'll never find or think of everything no matter how analytical your team is or how good your tools are. That's OK. Just go for the basics now. It doesn't take long to realize that the majority of Web application vulnerabilities are related to input validation, system configuration problems, and insiders abusing privileges they probably shouldn't have, including the following:

   - Cross-site scripting in search forms or message boards
   - SSL not being used or enforced throughout the application
   - Weak password requirements
   - Lack of account lockout after so many failed login attempts

- Informative authentication errors being returned to the user, resulting in username and password harvesting
- Weak mutual-factor authentication processes implemented per the Federal Financial Institutions Examination Council (FFIEC) requirements
- Session keys and cookies not expiring or being easily manipulated
- URL and/or form-field manipulation to bypass authentication or escalate privileges
- Sensitive information returned in server errors that can give an attacker a leg up on penetrating the system

Also, you may want to check out Microsoft's threat model called STRIDE that highlights the important areas of most applications:

Spoofing identity

Tampering with data

Repudiation

Information disclosure

Denial of service

Elevation of privilege

6. **Determine what's urgent and important** based on the likelihood and impact of each weakness. Always remember that not all security flaws are created equal. Focus first on the exploitable vulnerabilities in the most sensitive parts of the application or on the most sensitive systems. Some things will have a high likelihood but a low impact. Others may have a low likelihood with a high impact. Go for the items that are both high likelihood and high impact and work down from there.

   The DREAD model can be used here or you can develop your own in-house model for assessing risk. Either way, just keep your analysis consistent over time and across applications.

7. **Determine what can be done about each weakness** and when it can/will be resolved. As in step one, this involves setting goals and sticking to them. Your options are usually pretty simple: fix it (via code changes or other security controls) or obscure it in hopes that no one finds it. (I see that a lot, but it's not recommended long term.)

   Do you fix the problem(s) in the next release? Within six months? Never? You've got to be realistic based on how critical each item is and how much effort is involved. Also, be careful "fixing" security holes to the point that usability is affected. Aside from being hacked, the next worst thing that can happen is ticking off

your users to the point where they stop using the application or create new security holes working around stringent controls.

In essence, threat modeling is analyzing your Web application to find out what information flows where, outlining who can do what and when, and determining the worst that can happen. You can do all of this manually or you can use a software-based modeling tool such as Amenaza's SecurITree to help. If you have a large development team or a complex application, I recommend using a tool if you can. It can really speed up the process, and it looks pretty for the higher-ups to boot.

Given that threat modeling affects the entire development lifecycle, it's really something that needs to be done during the design phase if at all possible. So, now is probably a good time to get started. That said, don't let threat modeling drive your entire project or get in the way of your development efforts. I see all too often where developers and their managers obsess over this stuff to the point that it does more harm than good—especially at first. Don't drain the ocean and attempt to do everything possible to lock down your application's security. You'll just get in the way of yourself.

Instead, combine these techniques with some common sense and build out your threat modeling capabilities over the next few years and project iterations. It won't fix everything at once, but this one-bite-at-a-time approach will help get more people on board and allow your team to ease into the techniques and malicious mindset needed for effective threat modeling. In turn, you'll build better processes and bake security in up front so you don't have to worry about it as much in the future.

Resources from thawte

The Shortcut Guide to Managing Certificate Lifecycles

Extended Validation SSL Certificates: A NEW STANDARD TO INSPIRE TRUST, Improve Confidence and Increase Sales

About thawte

thawte is a leading global Certification Authority which offers a complete range of digital certificate products adds the elements of trust, integrity and privacy to all forms of information in transit over the internet, ensuring protection and peace of mind. thawte digital certificates interoperate smoothly with the most common web servers and browsers, so you can rest assured that you purchase of a thawte Digital Certificate will give your customers confidence in the integrity of your systems.

http://www.thawte.com